Simulink 기반
자동차 부품 연계보안 기술 연구

팀명: AutoShield
지도교수: 손준영

팀원: 202029178 홍재왕
202055627 레퐁푸
202055555 석재영

목 차

Ⅰ. 과제 배경 및 목표

1. 과제 배경

현대 자동차는 자율주행과 커넥티드카 기술로 ECU와 센서가 CAN, Ethernet으로 통신하며, CAN의 낮은 보안성과 Ethernet의 외부 노출로 해킹, 스푸핑 위협에 취약하다. 본 과제는 Simulink로 전장부품과 통신을 모델링하고, AES, SPECK 등 보안 알고리즘을 구현해 안전성을 검증한다.

2. 중간 과제 세부 목표

본 과제의 최종 목표는 Simulink를 활용하여 자동차 전장부품 간 통신 환경을 시뮬레이션하고, 보안 알고리즘을 적용해 실시간성과 보안성을 검증하는 체계적인 프레임워크를 개발하는 것이다. 국가보안기술연구소의 피드백을 반영하여, 최종 목표와 세부 목표 간 연관성을 명확히 하고, 구체적인 결과물을 도출하도록 수정했다. 그리고, 아직은 자동차 전장부품 간 통신 환경 제작에 미흡하다고 판단하여, Mathworks에서 제공하는 예시 자료를 토대로 진행하였다.

세부 목표는 다음과 같다:

가. Simulink 기반 코딩 및 테스트 환경 구축: 피드백을 반영해 착수보고서의 목표를 명확화하고, Simulink 환경에서 차량 통신 시뮬레이션과 보안 알고리즘 테스트를 위한 환경을 구축한다. Vehicle Network Toolbox를 활용해 ECU, 센서 간 데이터 흐름을 구현하며, MATLAB Function 블록으로 암호화 및 인증 알고리즘을 통합한다. 이는 보안 기능의 실시간 성능 평가를 가능하게 한다.

나. 차량 통신 시뮬레이션: Simulink의 예제(Gauge, Half Gauge, Slider, Lamp)를 참고하여 전장부품(가속도/속도 계기판, 브레이크 등)의 데이터 흐름을 모델링한다. 각 부품 간 메시지 흐름을 시뮬레이션하여, 예를 들어 브레이크 신호가 CAN 네트워크를 통해 어떻게 전달되는지 분석한다. 이는 보안 알고리즘 적용 시 통신 지연과 무결성을 평가하는 기초를 제공한다.

다. CAN 프로토콜 보안성 강화: CAN 프로토콜의 메시지 구조와 취약점(스푸핑, 리플레이)을 조사하고, 메시지 인증 코드(MAC)와 XOR 기반 경량 알고리즘을 구현해 유효한 메시지만 전달되도록 한다. 현재 초기 일부 구현 완료되었다.

Ⅱ. 요구사항 및 제약사항 분석에 대한 수정 사항

1. 대상 문제

현대 자동차의 ECU 네트워크는 CAN과 Ethernet을 통해 데이터를 교환하지만, CAN의 인증 및 암호화 부재로 스푸핑과 리플레이 공격에 취약하다. 예: 악의적 메시지 주입으로 브레이크 제어 오류 발생. Ethernet 기반 V2X 통신은 외부 해킹(예: 데이터 변조)에 노출된다. 센서(예: LiDAR)의 데이터 조작은 차량 오작동을 유발하며, 이는 운전자 안전과 차량 신뢰성을 위협한다.

2. 요구 조건

가. Simulink에서 전장부품과 통신 환경을 모델링하여 실제 차량 동작 시뮬레이션. ➔ CAN의 8바이트 데이터 제한과 ECU의 메모리 제약(예: 128KB)을 고려한 모델링. Simulink의 실시간 시뮬레이션 모드로 실제 동작과 유사성을 높인다.

나. AES-128, SPECK, HMAC을 구현하여 기밀성과 무결성을 보장. ➔ 경량 알고리즘을 우선 적용하고, SHA-256 해시 알고리즘을 추가해 메시지 인증 강화. 실제 ECU의 낮은 연산 능력을 반영해 알고리즘 최적화.

다. 알고리즘의 연산 부하, 처리 속도, 공격 저항력을 비교. ➔ Simulink 내의 간단한 기능 및 블록들을 활용하여, 단순 비교. 실제 하드웨어 테스트와의 오차를 줄이기 위해 시뮬레이션 파라미터 조정.

라. MATLAB App Designer로 알고리즘 선택 및 결과 시각화. ➔ UI를 간소화하여 알고리즘 선택과 성능 비교(그래프, 표)만 포함. MATLAB App Designer 대신 Python Dash를 추가 검토해 배포 용이성 확보.

마. 경량화된 알고리즘과 모듈화된 구현. ➔ 부품별 제약(예: 센서의 32KB 메모리)을 고려한 모듈별 코드 분리. Simulink의 Subsystem 재사용성을 높여 확장성 보장.

3. 제약사항

가. 제한된 하드웨어 리소스: ECU(예: 128KB 메모리, 8비트 프로세서)의 낮은 연산 능력은 AES-256 구현 시 지연(>10ms)을 초래한다. ➔ SPECK, ChaCha 등 경량 알고리즘을 우선 구현하고, Simulink로 리소스 사용(메모리 < 50KB) 시뮬레이션

후 최적 알고리즘 선정.

나. 통신 프로토콜 통합: CAN의 8바이트 데이터 제한은 보안 데이터 추가를 어렵게 하며, Ethernet의 복잡한 TLS 구현은 Simulink에서 시간이 소요된다. ➔ Vehicle Network Toolbox로 CAN 메시지에 MAC 추가, Ethernet은 TLS 간소화 버전 테스트. (CAN 대역폭 한계(8바이트)를 구체화하고, TLS 구현의 현실적 어려움을 반영

다. Simulink 숙련도 부족: Simulink의 복잡한 블록 설정(예: Vehicle Network Toolbox)은 초보자가 ECU 모델링에 2-3주 소요. ➔ MathWorks 튜토리얼(예: CAN Communication Example)로 학습하고, 간단한 ECU 모델부터 구현.

라. 알고리즘 성능 평가: Simulink에서 스푸핑, DoS 공격 재현은 실제 ECU 성능(예: 클럭 속도)과 오차 발생. ➔ 스푸핑, 리플레이 테스트 케이스 설계 후, MATLAB으로 연산 시간(<10ms)과 메모리 사용 분석.

마. UI 개발 시간: MATLAB App Designer 학습은 UI 개발에 2주 이상 소요. ➔ MATLAB App Designer로 간단한 알고리즘 선택 UI 개발, Python Dash 검토.

Ⅲ. 설계 상세화 및 변경 내역

Ⅳ. 과제 수행 내용 및 중간 결과(구성원별 개발 진척도)

1. 홍재왕

가. 보안 검증 로직(Security Verification) 설계 및 구현:

(1) Speed controller ECU 내 수신부에 대해 CAN 메시지의 무결성 및 인증 검증 로직을 설계하였다.

(2) CAN 수신 메시지의 msg.Data 필드에서 SpeedCommand, Counter, HMAC를 추출하여 이를 기반으로 메시지 유효성을 판단하는 SecurityVerification 함수를 작성하였다.

(3) 수신된 메시지의 무결성을 위해 SHA-256 기반 **HMAC 생성 함수 (compute_hmac_sha256)**를 직접 구현하였으며, Secret Key와 메시지 내용(SpeedCommand + Counter)을 이용하여 기대 HMAC을 계산하도

록 하였다.

(4) Simulink 내 MATLAB Function 블록을 활용해 SecurityVerification 모듈을 작성하고, HMAC 일치 여부와 Counter 증가 여부를 동시 만족해야만 isValid = true가 되도록 구현하였다.

```
function [isValid, cleanedData] = SecurityVerification(msg)
%#codegen

% 기본 출력 초기화
isValid = false;
cleanedData = uint8(0);

% Data 필드 추출
data = msg.Data;

% 데이터 길이 체크
if length(data) < 34
    return;
end

% 메시지 분리
speed_command = data(1);
counter = data(2);
received_hmac = data(3:34);  % 32바이트

% 예시용 키 (HMAC secret key)
key = uint8(1:16);  % 임시 키. 실제 시스템에서는 보안적으로 안전한 키 사용해야 함

% 인증 대상 데이터 구성
auth_data = uint8([speed_command, counter]);

% HMAC 계산
expected_hmac = compute_hmac_sha256(key, auth_data);

% Counter 상태 저장 (재전송 방지용)
persistent lastCounter
if isempty(lastCounter)
    lastCounter = uint8(0);  % 또는 255로 초기화해도 됨
end

% 검증
if isequal(received_hmac, expected_hmac) && counter > lastCounter
    isValid = true;
    cleanedData = speed_command;
    lastCounter = counter;
end
```

IMG 1. Security Verification Code

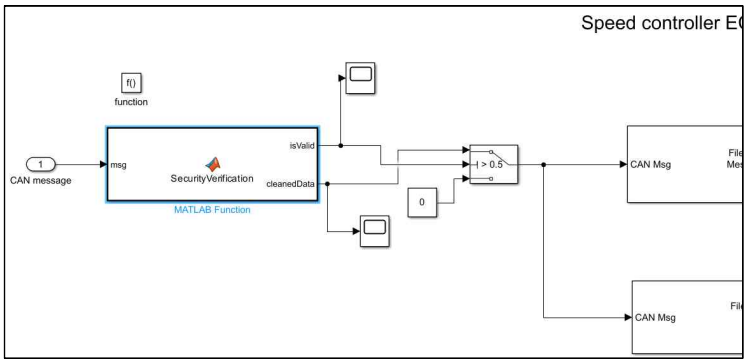
```
function hmac = compute_hmac_sha256(key, data)
% SHA-256 기반 HMAC 계산 함수
% 입력: key (uint8), data (uint8)
% 출력: hmac (uint8 vector, 32바이트)

import java.security.*;
import javax.crypto.*;
import javax.crypto.spec.*;

% Java HMAC 객체 생성
mac = Mac.getInstance('HmacSHA256');

% Key 초기화
keySpec = SecretKeySpec(uint8(key), 'HmacSHA256');
mac.init(keySpec);

% 데이터 처리
hmacJava = mac.doFinal(uint8(data));  % Java byte[]

% Java byte → MATLAB uint8
hmac = typecast(hmacJava, 'uint8');
```

IMG 2. Compute_Hmac_sha256.m

나. Counter 기반 재전송(Replay) 공격 방지 로직 설계:

(1) persistent 변수를 이용하여 lastCounter 값을 저장하고, 수신 메시지의 Counter 값이 lastCounter보다 클 경우에만 유효한 메시지로 간주하도록 하여 Replay 공격을 방어하고자 하였다.

(2) Counter 검증 결과는 isValid로 전달되며, 이는 Simulink의 Switch 블록과 연계되어 ECU 로직에 실제 데이터가 전달될지를 제어한다.



IMG 3. Compute_Hmac_sha256.m

다. Simulink 내 메시지 흐름 구조 개선:

(1) 기존 구조에서는 Speed controller ECU logic 내부의 CAN Unpack 블록들이 SecurityVerification 이전에 위치해 있었으나, **구조체 형태의 CAN 메시지(CAN_MESSAGE_BUS)**를 SecurityVerification 블록으로 직접 전달하고, 검증된 데이터만 Unpack 이후 ECU 로직에 전달되도록 구조를 수정하였다.

(2) Simulink의 데이터 타입 자동 추론 오류(Coder 오류, 구조체→비구조체 할당 오류 등)를 해결하기 위해 출력 포트의 Bus Object 정의, Signal Type 강제 지정, Message Width 고정 등의 방법을 적용하였다.

라. 결과 및 주요 한계점:

(1) 현재까지의 시뮬레이션 결과, isValid와 cleanedData 출력이 0으로 유지되는 현상이 발생하고 있어, HMAC 불일치 또는 메시지 구성 오류 가능성이 존재한다.

(2) 이로 인해 Speed controller에 올바른 SpeedCommand가 전달되지 않으며, 검증 로직이 정상적으로 메시지를 통과시키지 못하는 상태이다.

(3) 이후 디버깅을 위해 Display, Scope, Simulation Data Inspector를 활용

해 msg.Data 실시간 분석, HMAC 계산 결과 비교 검증을 진행 중이다.

2. 석재영

가. CAN 통신 환경 구축 및 설정 시도:

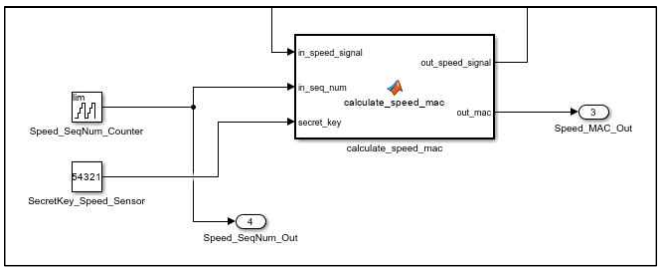
(1) ECU, 센서, 액추에이터의 Simulink 모델을 구성하고, Mathwork의 예제를 활용하여 CAN Pack 및 CAN Unpack 블록을 통해 RPM 및 Speed 신호의 CAN 메시지 송수신 인터페이스를 구축하였다.

(2) DBC 파일을 기반으로 CAN Pack의 Factor 설정 오류 및 CAN Unpack의 Extended ID 입력 형식 오류 등 CAN 통신 블록 설정 문제를 해결하고자 시도하였다.

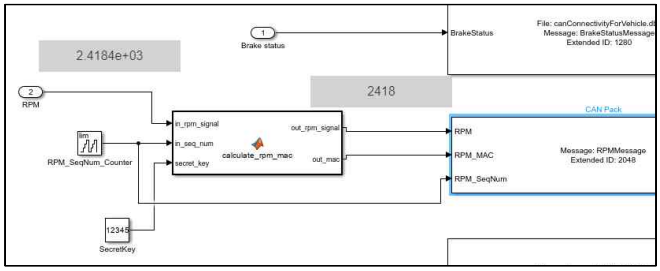
(3) MATLAB Function 및 Unit Delay 블록을 포함한 모델 내 주요 신호의 데이터 타입을 uint8, uint16, uint32 등으로 맞추어, 초기 '데이터 타입 불일치 오류'를 해결하고자 노력하였다.

나. MAC (메시지 인증 코드) 알고리즘 구현 및 검증 시도:

(1) MathWorks 예제를 참고하여 송신 측(calculate_speed_mac, calculate_rpm_mac)에서 RPM 및 Speed 신호에 대한 MAC을 생성하는 로직을 구현하였다.

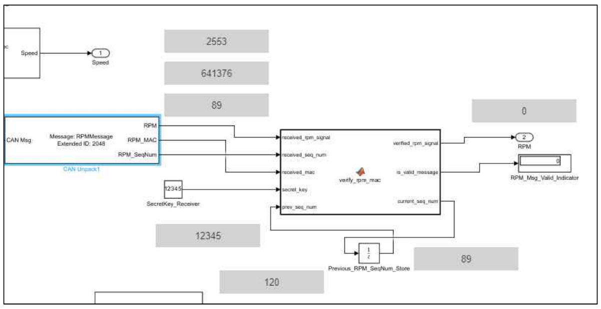


IMG 4. SpeedSensor

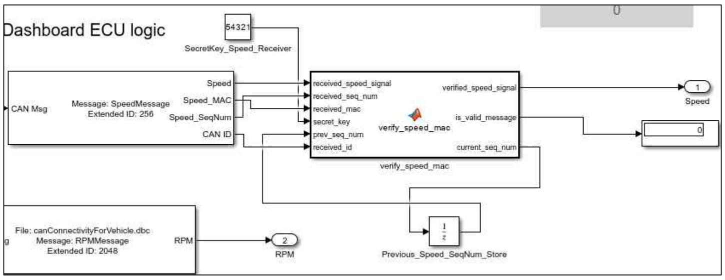


IMG 5. CalculateRPM

(2) 수신 측(verify_speed_mac, verify_rpm_mac)에서 수신된 MAC의 유효성을 검증하는 로직을 구현하여 'MAC Mismatch' 오류를 해결하고자 시도하였다.



IMG 6. VerifyRPM



IMG 7. VerifySpeed

(3) 송수신 간 secret_key를 일치시켜 보안 알고리즘 신뢰성을 확보하고자 하였다.

다. 시퀀스 넘버 기반 리플레이 공격 방어 구현 및 검증 시도:

(1) RPM 및 Speed 신호에 시퀀스 넘버를 포함하여 전송하고, 수신 측에서 prev_seq_num과 비교를 통해 시퀀스 넘버의 유효성을 검증하는 로직을 구현하였다. 'Replay Attack Suspected: SeqNum Mismatch' 오류를 해결하기 위해 노력하였다.

```
[Security Alert] RPMMessage - Replay Attack Suspected: SeqNum Mismatch (Received: 134, Previous: 134)
[Security Alert] RPMMessage - Replay Attack Suspected: SeqNum Mismatch (Received: 74, Previous: 134)
[Security Alert] RPMMessage - Replay Attack Suspected: SeqNum Mismatch (Received: 74, Previous: 74)
[Security Alert] RPMMessage - Replay Attack Suspected: SeqNum Mismatch (Received: 74, Previous: 74)
[Security Alert] RPMMessage - Replay Attack Suspected: SeqNum Mismatch (Received: 74, Previous: 74)
[Security Alert] RPMMessage - Replay Attack Suspected: SeqNum Mismatch (Received: 74, Previous: 74)
[Security Alert] RPMMessage - Replay Attack Suspected: SeqNum Mismatch (Received: 249, Previous: 249)
```

IMG 8. RPMMismatch

```
[Security Alert] SpeedMessage - Replay Attack Suspected: SeqNum Mismatch (Received: 27, Previous: 255)
[Security Alert] SpeedMessage - Message Tampering Suspected: MAC Mismatch (Received: 45610, Expected: 54570)
[Security Alert] SpeedMessage - Replay Attack Suspected: SeqNum Mismatch (Received: 27, Previous: 255)
[Security Alert] SpeedMessage - Message Tampering Suspected: MAC Mismatch (Received: 45610, Expected: 54570)
[Security Alert] SpeedMessage - Replay Attack Suspected: SeqNum Mismatch (Received: 27, Previous: 255)
[Security Alert] SpeedMessage - Message Tampering Suspected: MAC Mismatch (Received: 45610, Expected: 54570)
```

IMG 9. SpeedMismatch

(2) Unit Delay 블록을 활용하여 이전 시퀀스 넘버를 올바르게 저장하고 업데이트하는 메커니즘을 구축하고자 하였다.

라. CAN ID 기반 메시지 유효성 검증 시도:

(1) CAN Unpack 블록에서 출력되는 ID를 활용하여 MATLAB Function 내에서 CAN 메시지의 ID 유효성을 검증하는 로직을 구현하였으며, 'ID Mismatch' 오류를 해결하고자 시도하였다.

마. 결과 및 주요 한계점:

(1) 최종 시뮬레이션 결과, RPM 및 Speed 신호가 모두 0으로 표기되어 보안 알고리즘의 최종 검증에 성공하지 못했다**.** 이는 및 블록의 출력이 지속적으로 False를 반환하였음을 의미한다.

3. 레퐁푸

가. Dashboard ECU 예제 기반 Simulink 모델 분석 및 Brake ECU 설계:

(1) **MathWorks의 공식 예제 "Implement Distributed ECU Network over CAN"**을 바탕으로, Dashboard ECU의 하드웨어 추상화 구조 및 신호 흐름을 분석하였다.

(2) 해당 예제를 통해 다음과 같은 Simulink의 사용법을 학습하였다:

ㄱ. Sine Wave 블록을 활용한 테스트 신호 생성

ㄴ. 데이터 타입 변환을 위한 Data Type Conversion, 범위 제한을 위한 Saturation 블록의 실사용

ㄷ. CAN Pack, CAN Unpack 블록 설정 방법 및 ID 할당 기준 이해

ㄹ. Gauge 및 Lamp 블록을 활용한 실시간 시각화 구성법

나. BrakeSensor ECU (송신부) 설계 및 구현:

(1) Brake Force 신호는 Sine Wave 블록으로 주기적으로 생성되며, Saturation 블록을 통해 물리적 범위(0~255)로 제한된 후, uint8 타입으로 변환되었다.

(2) CAN Pack 블록에서는 CAN ID 200을 사용하였고, BrakeForce 신호를 8

비트 단일 메시지로 패킹하였다.

(3) CAN Transmit 블록을 통해 MathWorks Virtual Channel 1에서 CAN 메시지를 전송하도록 설정하였다.

다. ABSECU (수신부) 구현 및 시각화:

(1) CAN Receive → CAN Unpack을 통해 BrakeForce 메시지를 성공적으로 수신 및 디코딩하였고, Gauge 블록을 통해 실시간으로 BrakeForce 수치를 시각화하였다.

(2) Compare To Constant 블록을 이용하여 BrakeForce 값이 220을 초과할 경우, Lamp 블록이 점등되도록 구성하여 비상 제동 시나리오에 대한 경고 메커니즘을 구현하였다.

다. 오류 처리 및 시뮬레이션 조정:

(1) CAN Configuration 블록에서 Virtual Channel 설정 오류, 데이터 타입 불일치 등 발생한 문제들을 실험적으로 해결하였으며, Simulink 실행 중 발생한 오류들(Fixed-step 설정 누락, Stop Time 설정 미비 등)은 시뮬레이션 파라미터 조정을 통해 수정하였다.

(2) Subsystem Label, Masking 기능을 적용하여 각 ECU별 기능을 명확히 시각화하였고, 신호 흐름 라벨링을 통해 전체 시스템 동작 구조를 가시화함으로써 모델의 이해도 및 유지보수성을 향상시켰다.

라. 결과 및 주요 한계점:

(1) 최종 시뮬레이션 결과, RPM 및 Speed 신호가 모두 0으로 표기되어 보안 알고리즘의 최종 검증에 성공하지 못했다**.** 이는 및 블록의 출력이 지속적으로 False를 반환하였음을 의미한다.

4. 결론 및 결과

현재 Simulink에서 ECU, 센서 모델링과 CAN 통신 초기 구현이 완료되었으며, MAC, SHA-256, XOR 기반 보안 로직과 리플레이 공격 방어 메커니즘이 설계되었다. 그러나 HMAC 불일치, 데이터 타입 오류 등으로 신호 출력이 0으로 유지되어 보안 검증이 미완성이다. 디버깅을 위해 Scope와 Simulation Data Inspector를 활용 중이며, 하드웨어 제약(예: 8바이트 CAN 제한)을 고려한 최적화가 필요하다.

V. 향후 진행 계획

1. 개발 일정

아래 표는 2025년 7월 3-4주차부터의 향후 진행 계획을 기존 일정과 피드백을 반영하여 수정한 것이다. 목표는 Simulink 기반 차량 통신 프레임워크와 보안 알고리즘 검증이며, 하드웨어 제약과 실시간성을 고려한다.

기간	주요 작업	세부 작업	담당자
7월 3-4주 (7.19-7.31)	보안 알고리즘 구현	- CAN 통신에 MAC, XOR, SHA-256 구현 완료 - Ethernet 통신에 TLS 간소화 버전 테스트 - SPECK 알고리즘 Simulink 통합	홍재왕 (CAN 보안) 석재영 (Ethernet 보안) 레퐁푸 (부품 모델링 지원)
8월 1-4주 (8.1-8.31)	통신 시뮬레이션 및 UI 개발	- Simulink 예제(Gauge, Slider)로 부품별 메시지 흐름 최적화 - 스푸핑, DoS 공격 시나리오 테스트 - MATLAB App Designer로 UI(알고리즘 선택, 성능 그래프) 완성	레퐁푸 (UI, 모델링) 홍재왕, 석재영 (테스트)
9월 1-2주 (9.1-9.19)	성능 테스트 및 보고서 작성	- 알고리즘 성능 비교(지연 < 10ms, 메모리 < 50KB) - 하드웨어-in-the-loop(HIL) 테스트 준비 - 최종보고서 및 평가표 작성	홍재왕, 석재영, 레퐁푸
9월 3-4주 (9.20-9.30)	발표 준비 및 보완	- 발표 자료 제작 - Simulink 모델 및 UI 데모 시연 리허설	홍재왕, 석재영, 레퐁푸
10월 1-2주 (10.1-10.15)	결과물 업로드	- 최종 코드, 모델, 보고서 정리 - AWS S3에 업로드 및 문서화 완료	레퐁푸 (문서화) 홍재왕, 석재영 (검토)

TABLE1. 개발 일정 및 역할 분담

2. 역할 분담

구성원별 역할은 프로젝트 목표(자동차 전장부품 통신 프레임워크 개발 및 보안 알고리즘 검증)와 일정에 맞춰 다음과 같이 수정되었다:

가. 홍재왕 (CAN 프로토콜 담당): CAN 프로토콜의 메시지 구조와 취약점(스푸핑, 리플레이)을 분석하고, Simulink에서 CAN 통신 시뮬레이션을 구현한다. MAC, XOR, SHA-256을 적용해 유효 메시지 전송을 보장하며, 초기 구현을 완료한다. 보안 알고리즘 성능 비교(지연 시간, 메모리 사용)에 참여한다.

나. 석재영 (TCP/IP, Ethernet 담당): Automotive Ethernet과 TCP/IP의 통신 특성과

V2X 보안 요구사항을 분석한다. Simulink에서 Ethernet 통신 시뮬레이션을 구현하고, TLS 간소화 버전 적용을 테스트한다. 알고리즘 성능 비교와 데이터 흐름 분석을 지원한다.

다. 레퐁푸 (Simulink 담당): Simulink 환경을 구축하고, MathWorks 예제(Gauge, Slider)를 활용해 ECU, 센서, 계기판의 메시지 흐름을 모델링한다. 보안 알고리즘 통합 워크플로우를 설계하고, MATLAB App Designer로 UI(알고리즘 선택, 성능 시각화)를 개발한다. 보고서 작성과 문서화를 주도한다.

기존 역할에서 해시 알고리즘(SHA-256), 차량 통신 흐름(Gauge, Slider), 초기 CAN 보안 일부 구현 완료를 반영했다. LLM은 국가보안기술연구소 피드백에 따라 제외했으며, UI와 성능 비교를 구체화했다. 역할은 유연하게 조정 가능하다.

VI. 참고문헌

MathWorks. (2025). Simulink Documentation: Vehicle Network Toolbox. [Online] Available: https://www.mathworks.com/help/simulink/

Bosch. (1991). CAN Specification 2.0. Robert Bosch GmbH.

Miller, C., & Valasek, C. (2015). Remote Exploitation of an Unaltered Passenger Vehicle. Black Hat USA.

MathWorks. (2025). Simulink for Automotive Applications. [Online] Available: https://www.youtube.com/@MATLAB

HuggingFace. (2025). Transformers Documentation. [Online] Available: https://huggingface.co/docs/transformers/